CSE 3500 Algorithms and Complexity – Fall 2016
Lecture 8: September 22, 2016

Master Theorem

- In the last lecture we showed how to use the repeated substitutions technique to solve a recurrence relation. In this lecture we study the Master theorem.
- The Master theorem considers a recurrence relation of the kind:

$$T(n) = \begin{cases} c & \text{if } n \leq d \\ aT(n/b) + f(n) & \text{if } n > d \end{cases}$$

  where $a > 0, b > 1, c$, and d are integer constants, and $f(n)$ is a non-negative integer function of n.

- There are three cases to consider and in each case we compare $n^{\log_b a}$ with $f(n)$.
- **Case 1:** $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$. In this case the solution is: $T(n) = \Theta(n^{\log_b a})$.
- **Case 2:** $n^{\log_b a} = \Theta(f(n))$. In this case the solution is: $T(n) = \Theta(f(n) \log n)$.
- **Case 3:** $n^{\log_b a} = O(f(n)/n^{\epsilon})$ for some constant $\epsilon > 0$ and there exists a constant $q < 1$ such that $af(n/b) \leq qf(n)$. In this case, $T(n) = \Theta(f(n))$.

Example 1

- The recurrence relation for binary search we obtained was:

$$T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ T(n/2) + 1 & \text{if } n > 1 \end{cases}$$

- For this recurrence relation $a = 1, b = 2$ and $f(n) = 1$. $n^{\log_b a} = n^{\log_2 1} = 1$. Thus, $f(n) = \Theta(n^{\log_b a}$. As a result, case 2 holds and we infer that $T(n) = \Theta(f(n) \log n) = \Theta(\log n)$.

Example 2

- The recurrence relation for merge sort is:

$$T(n) = \begin{cases} 1 & \text{if } n \leq 2 \\ 2T(n/2) + n & \text{if } n > 2 \end{cases}$$

- For this recurrence relation $a = 2, b = 2$ and $f(n) = n$. $n^{\log_b a} = n^{\log_2 2} = n$. Thus, $f(n) = \Theta(n^{\log_b a}$. As a result, case 2 holds and we infer that $T(n) = \Theta(f(n) \log n) = \Theta(n \log n)$.

Example 3

- Consider the following recurrence relation:

$$T(n) = \begin{cases} 1 & \text{if } n \leq 2 \\ 5T(n/2) + n^2 & \text{if } n > 2 \end{cases}$$

- For this recurrence relation $a = 5, b = 2$ and $f(n) = n^2$. $n^{\log_b a} = n^{\log_2 5}$. Note that $\log_2 5 > 2.32$. Thus, for a value of $\epsilon = 0.3$, $f(n) = O(n^{\log_b a - \epsilon})$. This means that case 1 holds and hence $T(n) = \Theta(n^{\log_2 5})$. This is what we got using the repeated substitutions technique as well. Note that the choice of ϵ is not unique. For example, we could have chosen ϵ to be 0.2.

Example 4

- Now consider the following recurrence relation:

$$T(n) = \begin{cases} 1 & \text{if } n \leq 2 \\ 12T(n/3) + n^3 & \text{if } n > 2 \end{cases}$$

- For this recurrence relation $a = 12, b = 3$ and $f(n) = n^3$. $n^{\log_b a} = n^{\log_3 12}$. Note that $\log_3 12 < 2.3$. Thus, for a choice of $\epsilon = 0.5$, $n^{\log_b a} = O\left(\frac{f(n)}{n^{0.5}}\right)$. Also, $a\ f(n/b) = 12(n/3)^3 = (12/27)n^3$. In other words, $a\ f(n/b) \leq qf(n)$ for $q = (12/27)$ which is less than one. As a result, case 3 holds. Therefore, we conclude that $T(n) = \Theta(n^3)$.

Quick sort

- Input: $X = k_1, k_2, \ldots, k_n$; Output: Sorted X.

- The quick sort algorithm employs divide-and-conquer and works as follows:

QuickSort(X)
if $n = 1$ **then** quit;
if $n = 2$ **then**
{**if** $k_1 > k_2$ **then** swap k_1 and k_2; quit;}
Pick a pivot element k from X;
Partition X into X_1 and X_2 using k as follows:
$X_1 = \{q \in X : q < k\}$ and $X_2 = \{q \in X : q > k\}$;
Recursively sort X_1 to get Y_1;
Recursively sort X_2 to get Y_2;
Output Y_1, k, Y_2;

Analysis of quick sort

- Let $T(n)$ be the run time of quick sort on any input of size n.
- Given that partitioning takes n comparisons, we see that: $T(n) = T(|X_1|)+T(|X_2|)+n$.
- One of the worst cases happens when the input is already in sorted order. In this case, whenever a recursive call is made, one of the two parts is empty.
- The recurrence relation for $T(n)$ corresponding to the above worst case is: $T(n) = T(n-1) + n = T(n-2) + (n-1) + n = T(n-3) + (n-2) + (n-1) + n = \cdots = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2)$.
- Consider the possibility that whenever a recursive call is made, both X_1 and X_2 are of the same size. In this case the recurrence relation for $T(n)$ will become: $T(n) = 2T\left(\frac{n}{2}\right)+n$. This solves to: $T(n) = \Theta(n \log n)$ (using the Master theorem, for example). This is one of the best cases.
- In the next lecture we will show that the expected run time of quick sort is $O(n \log n)$.